RELATORIO
PROGRAMAÇÃO PARALELA

PROFESSOR: DANIEL WEINGAERTNER
ALUNOS:
GUSTAVO TOSHI KOMURA GRR20102342
LUIS ALEXANDRE DESCHAMPS BRANDÃO GRR20085461

Solução

A estrutura utilizada consiste de 4 double que são:

- x e y: valores de x e y no plano cartesiano.
- fxy: valor do resultado da função f(x,y).
- “valor”: valor do resultado de cada iteração calculados a partir da função de atualização.

Os valores de x e y, são determinados pela divisão da quantidade de intervalos determinados pelos valores nx e ny, que são passados por parâmetro que devem estar nos limites [0,2] para x e [0,1] para ny.

O valor de fxy é determinado pela função f(x,y) = 4π^2sin(2πx)sinh(2πy).

O valor de “valor” é determinado pela função
u(x,y) = 1/((2/hx^2)+(2/hy^2)+k^2)(f(x,y) + (1/hx^2)(u[x-1,y] + u[x+1,y]) + (1/hy^2)(u[x,y-1] + u[x,y+1])).

A malha (plano cartesiano) foi organizada em uma matriz de estruturas que ao ser inicializada são calculados os valores de x, y e fxy baseando-se nos valores de entrada nx e ny, e o “valor” de cada ponto é inicializado com “0” por causa da solução inicial.
Ao se executar o programa é verificado se todos os parâmetros foram passados e se eles estão corretos, depois é verificado qual o método de solução escolhido (Jacobi ou Red-Black Gaussian)
Para o método Red-Black Gaussian, inicialmente é armazenado o tempo atual para depois se fazer o cálculo do tempo de execução, e em seguida é criada uma matriz (nx + 1)x(ny + 1) e são executados n iterações, que é o valor passadas por parâmetro, onde em cada iteração existem dois loops paralelos, o primeiro calcula a função de atualização nos pontos red e o segundo nos pontos black. Por fim depois de passar por todas as iterações é pego o tempo atual e depois é feito o cálculo do tempo de execução, e no final é calculado o resíduo e a solução é gravada em um arquivo.

Tentativas de melhoramento

1 - Trocar o uso de uma matriz por um vetor, no entanto o tempo de execução foi maior, ou seja o desempenho caiu.

2 - Ao invés de armazenar o valor de fxy fazer o calculo a cada iteração, no entanto o tempo de execução foi na maior parte das vezes mais que o dobro do tempo. Entretanto, nesse caso, o problema escalava de maneira adequada; quando dobrava-se o numero de threads o tempo de processamento era dividido por dois.

3 - Implementar Red-Black Gaussian sobre dois vetores, um vetor representado os pontos red e o outro os pontos black, desta forma um vetor seria somente para leitura e outro somente para escrita. O problema desta implementação tem relação com a tentativa número 1, ou seja, trocar vetor por matriz deixa o programa mais lento e também por causa da dependência de dados, pois por exemplo suponha que estamos calculando um ponto red e para calcular este ponto na função de atualização precisamos do valor de fxy e este valor é armazenado na estrutura, no entanto esta estrutura está no vetor red, lugar onde deveríamos somente escrever e não ler. Deste modo para evitar este problema seria possível fazer o cálculo de fxy a cada iteração apesar do problema da tentativa 2, no entanto continuamos com a

dependência de dados, pois para calcular fxy do ponto red precisamos dos valores de x e y que também estão na estrutura localizada no vetor red.

Resultados:

A tabela *Tabela 1 - Resultado Testes* e os gráficos *Gráfico 1 - Resultados de Testes - matriz 32 33*, *Gráfico 2 - Resultados Testes - matriz 1024 1025* e o *Gráfico 3 - resultados Testes - matriz 2048 2049* mostram os resultados da execução do programa. Todos os gráficos trazem a média do tempo da execução com 100 e 30 rodadas de testes sendo que com 100 testes foram executados numa máquina pessoal que possui um FX-8230 (8 cores, 4.00 GHZ) e 30 rodadas foram executados na máquina latrappe que possui um Intel(R) Xeon(R) CPU E5-2680 v2 @ (20 cores 2.80GHz)

Tabela 1 - Resultado Testes

Cotes/TamMatriz	32x32 100	32x32 30	33x33 100	33x33 30
1	0.027406	0.039644	0.033482	0.060191
2	0.038915	0.042674	0.045864	0.055498
4	0.097017	0.090191	0.103841	0.095914
8	0.182379	0.140244	0.183176	0.142951

	1024x1024 100	1024x1024 30	1025x1025 100	1025x1025 30	2048x2048 100	2048x2048 30	2049x2049 100	2049x2049 30
1	27.502183	31.178097	28.449898	45.422426	113.814782	122.802597	114.768726	171.380766
2	26.832612	14.978149	24.993057	23.897554	109.462931	59.220685	101.278759	102.041294
4	21.823649	8.532282	20.789515	12.80174	86.743838	32.263015	86.46898	45.09318
8	20.224512	8.982852	20.60025	11.228148	80.404778	21.964727	81.347651	25.397213

Gráfico 1 - Resultados de Testes - matriz 32 33

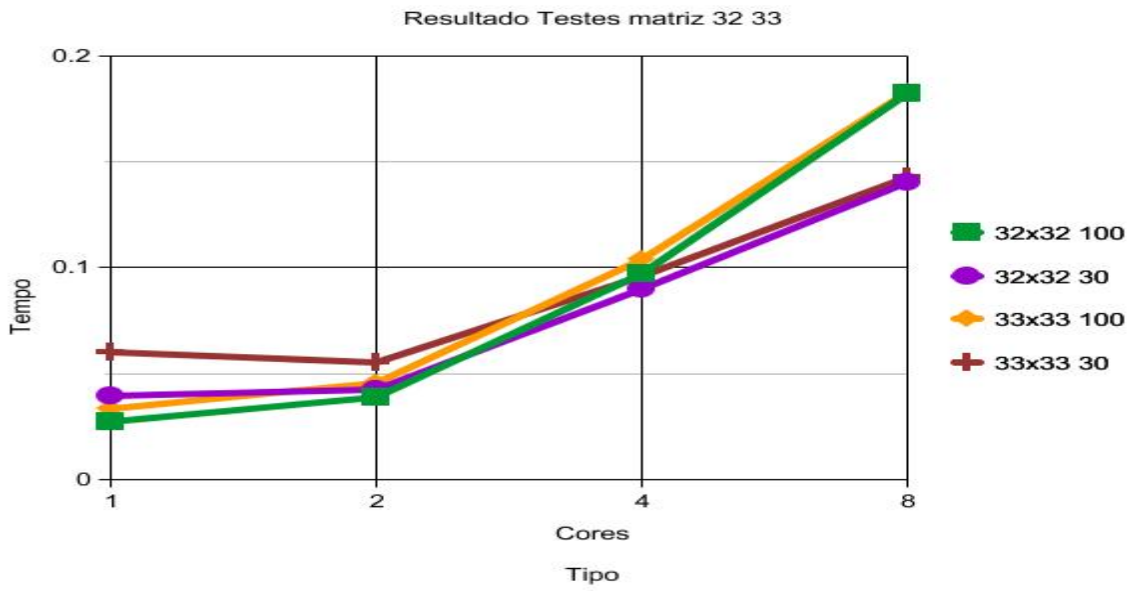
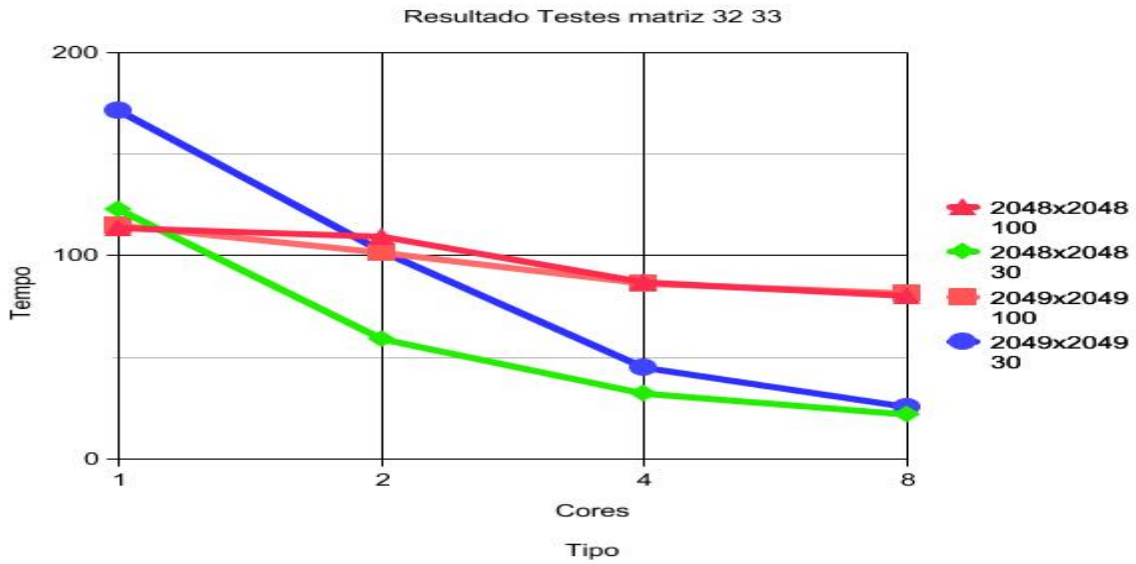


Gráfico 2 - Resultados Testes - matriz 1024 1025

Gráfico 3 - Resultados Testes - matriz 2048 2049

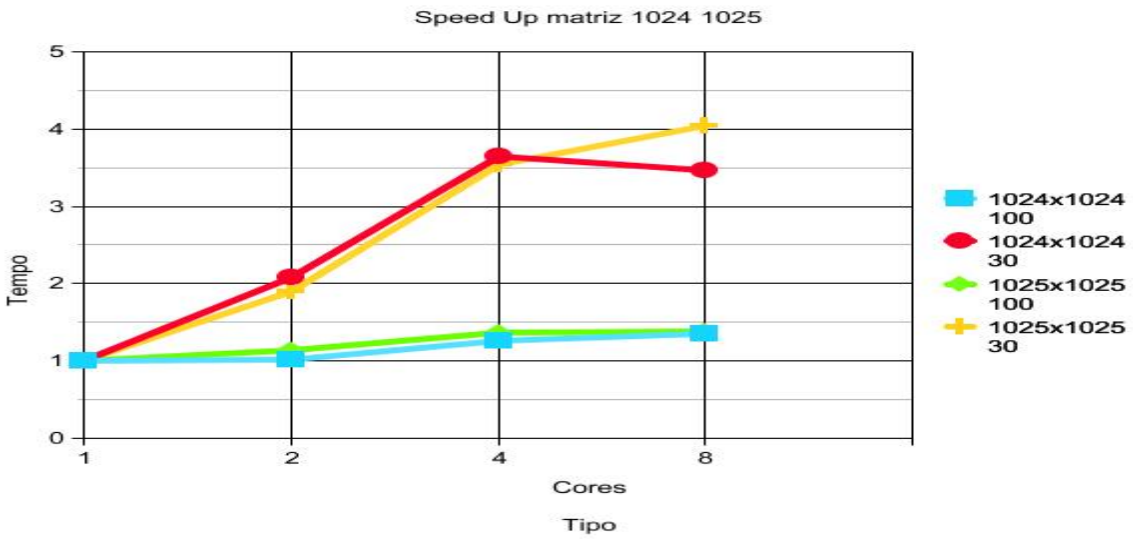


Speed Up

Cotes/TamMatriz	32x32 100	32x32 30	33x33 100	33x33 30
1	1	1	1	1
2	0.70425285879	0.92899657871	0.045864	1.08456160582
4	0.28248657452	0.43955605326	0.3224352616	0.62755176512
8	0.15026949375	0.28267875988	0.18278595449	0.42106036334

	1024x1024 100	1024x1024 30	1025x1025 100	1025x1025 30
1	1	1	1	1
2	1.02495362732	2.08157209546	1.13831205202	1.90071444132
4	1.26020094073	3.65413344285	1.3684733867	3.54814470533
8	1.35984408425	3.47084611881	1.38104624944	4.0454067759

	2048x2048 100	2048x2048 30	2049x2049 100	2049x2049 30
1	1	1	1	1
2	1.03975639022	2.07364364326	1.13319640893	1.67952364461
4	1.31207915887	3.80629637373	1.32728206115	3.8005917081
8	1.41552261981	5.59090021925	1.41084253312	6.74801467389



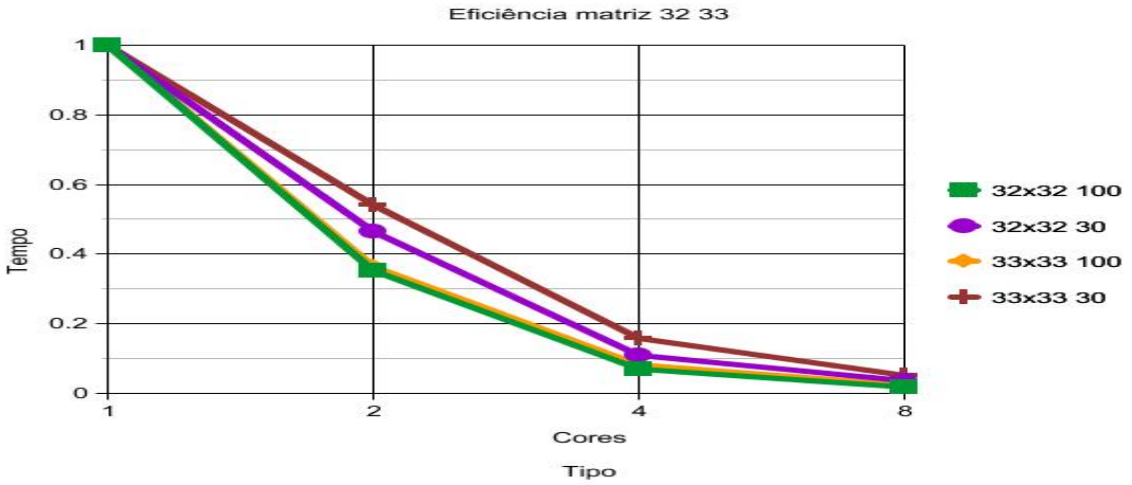
Eficiência

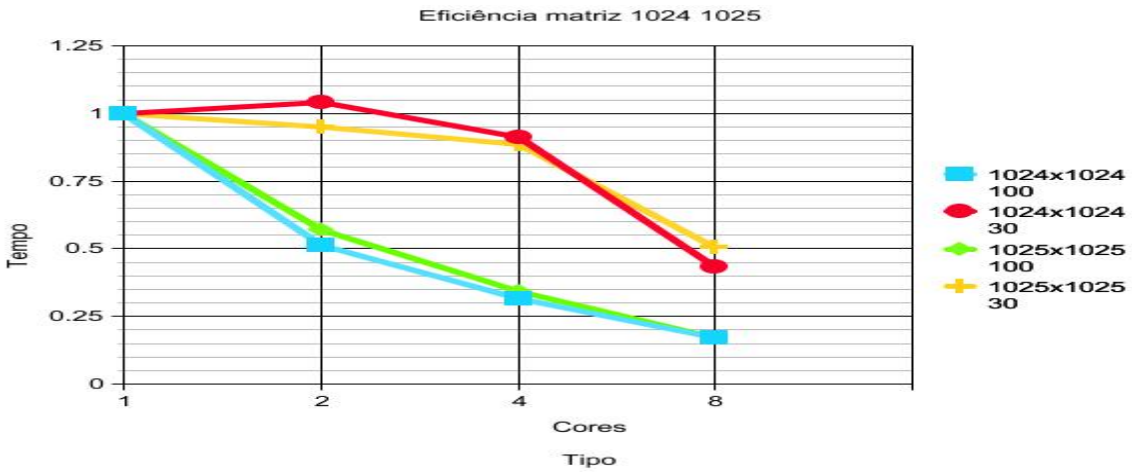
Cotes/TamMatriz	32x32 100	32x32 30	33x33 100	33x33 30
1	1	1	1	1
2	0.3521264294	0.46449828936	0.3650139543	0.54228080291
4	0.07062164363	0.10988901332	0.0806088154	0.15688794128
8	0.01878368672	0.03533484498	0.02284824431	0.05263254542

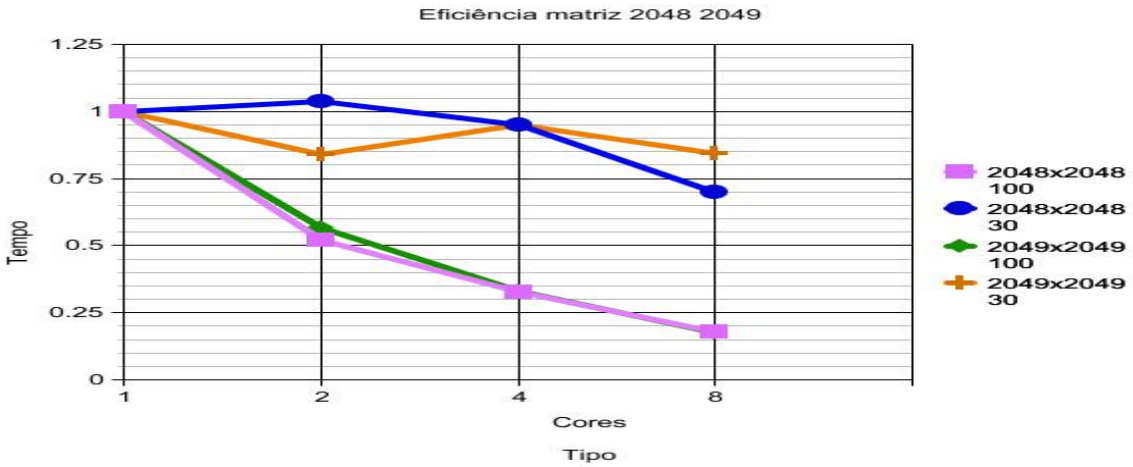
	1024x1024 100	1024x1024 30	1025x1025 100	1025x1025 30
1	1	1	1	1

2	0.51247681366	1.04078604773	0.56915602601	0.95035722066
4	0.31505023518	0.91353336071	0.34211834668	0.88703617633
8	0.16998051053	0.43385576485	0.17263078118	0.50567584699

	2048x2048 100	2048x2048 30	2049x2049 100	2049x2049 30
1	1	1	1	1
2	0.51987819511	1.03682182163	0.56659820447	0.83976182231
4	0.32801978972	0.95157409343	0.33182051529	0.95014792703
8	0.17694032748	0.69886252741	0.17635531664	0.84350183424







Observando os resultados podemos perceber que quando as matrizes possuem um tamanho pequeno ao se aumentar o número de threads o desempenho cai e isso se deve ao fato de que com o aumento de processadores, a comunicação aumenta. Como a quantidade de dados é pequena, o tempo gasto com a comunicar se torna mais relevante para o desempenho do que o tempo de processamento, ou seja, se gasta mais tempo comunicado do que processando.

Trabalhos futuros

Para trabalhos futuros poderíamos ganhar desempenho se quebrarmos a estrutura em duas, sendo uma estrutura armazenamento o valores de x, y e fxy e outra que possua somente o “valor”. Com esta implementação poderia haver um ganho, pois x, y e fxy seria somente para leitura após serem inicializados e desta forma poderíamos implementar a tentativa 3 sem nenhuma dependência de dados. Voltando ao exemplo de calcularmos a função de atualização em um ponto red, não teríamos nenhuma dependência de dados, pois os valores x, y e fxy somente seriam lidos da nova estrutura e os valores do stencil também somente seriam lidos das estruturas que estão no vetor black e desta forma só iriamos escrever na estrutura red.

Conclusão

Podemos perceber pelos resultados que com o aumento no número de threads a implementação paralela melhora o desempenho para matrizes de tamanhos grandes e isso se deve ao fato de que o processamento dos dados é bem distribuído entre as threads, no entanto para matrizes de tamanho pequeno o melhoramento do desempenho não foi alcançado, pois o tempo de comunicação tornou-se mais relevante que o tempo de processamento.